FPGA Gaussian Random Number Generators with Guaranteed Statistical Accuracy

David B. Thomas
Dept. of Electrical and Electronic Engineering
Imperial College
London, England
Email: dt10@imperial.ac.uk

***Abstract*—Many types of stochastic algorithms, such as Monte-Carlo simulations and Bit-Error-Rate testing, require very high run-times and are often trivially parallelisable, so are natural candidates for execution using FPGAs. However, the applications are reliant on Gaussian Random Number Generators (GRNGs) with good statistical properties, as very small biases over trillions of random samples can lead to incorrect results. Previous hardware GRNGs have focussed on area-efficient algorithms to produce Gaussian distributions under idealised assumptions, but do not make statements about the actual distribution coming out of real fixed-point hardware. In this paper, we present a new type of GRNG called a Piecewise-CLT, which uses a weighted blend of many small smooth distributions to approximate the Gaussian. By adjusting the weights, it is possible to directly target the distribution of the Gaussian, resulting in a circuit with an exactly quantified output distribution. Three members of the PwCLT family are presented, ranging from medium-area with good quality, up to a generator providing guaranteed statistical accuracy out to 12-sigma. We also show that PwCLT provides a better area-accuracy tradeoff than all existing high-speed scalar FPGA GRNGs, and can provide extremely high levels of statistical quality not possible in any previous methods.**

***Keywords*-Monte-Carlo; Gaussian; Random Numbers; RNG**

I. Introduction

The Gaussian distribution is one of the most important probability distributions, as it naturally arises in many situations due to the central limit theorem, with uses including the modelling of asset price returns in finance, noise in communications channels, and variance of propagation delay in VLSI processes. Although the Gaussian distribution is usually only an approximation to real-world phenomena, it often captures many of the required statistical features, so the Gaussian distribution has been embedded in many mathematical frameworks for modelling these problems.

Often it is necessary to resort to simulation of the systems because analytical solutions are infeasible; for example, in computational finance it is often computationally infeasible to numerically solve stochastic differential equations so Monte-Carlo solutions are required, while Bit-Error-Rate (BER) testing of communications codecs requires actual noise samples to run through algorithms. A common problem when simulating systems directly is raw computational complexity: Monte-Carlo solutions only converge with the square of the number of random trials, so investigating the performance of communications channels at the $\pm 8\sigma$ level requires around 10^{16} noise samples.

FPGAs are particularly attractive for these tasks, as they allow many simulations to execute in parallel using pipelining and spatial replication. In such hardware implementations a key concern is the efficient implementation of Gaussian Random Number Generators (GRNGs) – the GRNGs must be as small as possible to allow the greatest degree of parallelism, while providing enough statistical accuracy for simulations involving many trillions of random samples.

This paper develops a family of FPGA-optimised GRNGs which match or exceed existing FPGA GRNGs in terms of resource consumption and generation rate. However, unlike previous GRNGs the new family of generators is designed to allow for hard quantitative guarantees of distribution accuracy, which allows users to make an informed resource vs quality tradeoff based on the needs of their application. The proposed generators can also provide samples with guaranteed quality beyond the $\pm 10\sigma$ and $\pm 12\sigma$ ranges, corresponding to sample sizes of $10^{25}+$ and $10^{33}+$, which far exceeds the capability of any existing FPGA GRNG.

The key contributions of this paper are:

- A parametrisable cross-platform hardware architecture for GRNGs which can be configured to provide different resource-quality tradeoffs;
- A set of three specific parametrisations designed for generating samples in the $\pm 8\sigma$, $\pm 10\sigma$, and $\pm 12\sigma$ ranges, covering application needs from low-quality up to ultra-high quality;
- An evaluation of the exact statistical characteristics of the generators, measured in terms of the maximum relative CDF error, and empirically in terms of χ^2 tests;
- An open-source platform-independent distribution for the three parameter sets, allowing practitioners and academics to use the generators.

II. Background

The standardised continuous uni-variate Gaussian Probability Distribution (PDF) is defined as:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \tag{1}$$

with the Cumulative Distribution Function (CDF):

$$\Phi(x) = \int_{-\infty}^{x} \phi(t)dt = \frac{1}{2}\left(1 + \operatorname{erf}\left(x/\sqrt{2}\right)\right) \tag{2}$$

Gaussian distributions with different means and variances can be created through basic scaling and translation of the standardised distribution, so in this paper we will consider only the standardised version.

In software the continuous Gaussian distribution is the target distribution for GRNGs, as single-precision floating-point numbers can be considered a good approximation to continuous variates. However, in FPGA architectures it is common to use fixed-point numbers, with the number of fractional bits (w_f) being chosen to minimise resource usage while maintaining the numerical and statistical properties required in the target application. The Gaussian can be discretised in terms of the CDF at the mid-points between representable numbers [1], giving the discretised Probability Mass Function (PMF) and CDF:

$$\Pr[x = N_C] = \Phi(x + 2^{-w_f}/2) - \Phi(x - 2^{-w_f}/2) \quad (3)$$

$$\Pr[x \leq N_C] = \sum_{j=-\infty}^{2^{w_f} \times x} \Pr[2^{-w_f} \times j = N_C] \quad (4)$$

From an application developer's point of view the implementation of a GRNG is not that interesting – all that matters is that it meets certain requirements. Traditionally these are viewed in terms of area (resource usage), speed (how many samples are generated per second), but quality should also be considered. A common quality metric is the empirical χ^2 test, which buckets up batches of generated samples and compares them to the desired distribution. Such testing is important, but needs to be applied across a variety of bucketing strategies, and with extremely large sample sizes.

However, it is also important to look at the precise distribution of a generator in order to exactly quantify how the generator matches up against the discrete Gaussian distribution. A simulator may be run over many thousands of nodes for hundreds of hours, so any small sample test may not identify flaws found at a large scale. Some papers consider absolute PMF error as a metric, but this ignores two problems:

- The effect of statistical flaws is proportional to relative error, not absolute error – an absolute error of 10^{-9} may be very good close to the median, but the same error at 8σ it would mean the tail was catastrophically over-weighted.
- If a large segment of the PMF has a tiny absolute error in the same direction it can accumulate to a much larger (and detectable) error in the CDF for the same segment.

In this paper we consider relative CDF error as the gold-standard of generator quality, as it addresses both of these problems, and also has a natural statistical and intuitive interpretation. However, it is a difficult metric to target with existing techniques, which motivated the development of the new PwCLT architecture.

[1]In reality there are multiple valid definitions of "discretised Gaussian", both from a mathematical and practical point of view, but for space reasons we can only consider one here. However, the method proposed in this paper is general enough to target any of the competing definitions.

III. The PwCLT Algorithm

The approach to sampling from the Gaussian distribution used in this paper is to approximate a complex target distribution by sampling from a weighted mixture of much simpler distributions. Given a target PDF $t(x)$, the goal is to choose a set of n component distributions with PMFs $c_1(x)..c_n(x)$ and a set of non-negative weights $p_1(x)..p_n(x) \in [0, 1]$, such that the weighted combination $a(x)$ is close to the target PMF:

$$t(x) \approx a(x) = \sum_{i=1}^{n} p_i c_i(x), \quad \text{where } \sum_{i=1}^{n} p_i = 1 \quad (5)$$

Similarly the CDF of the approximation is the weighted sum of the component CDFs.

If we pre-suppose some method for cheaply sampling from the component distributions, sampling from the approximation has two stages:

1) Randomly select component distribution j, such that $\Pr[j = i] = p_i$.
2) Generate and return a random sample from component distribution c_j.

These two steps correspond to the two main architectural difficulties in designing a generator using mixtures, while Equation 5 presents the optimisation challenge:

Component selection: The weighted selection of distributions can be achieved using an alias-table [1], but the fixed-point approach used in existing work [2] is inefficient when approximating very small probabilities in the tails of distributions. As one of the goals of this paper is to support accurate relative CDF error far into the tails, a new approach for floating-point hardware alias tables is developed for this work in Section III-A.

Component distribution: There are many possible choices for component distributions, with a tradeoff between flexibility and resource usage, but this paper selects overlapping components derived from the central limit theorem, explained in Section III-B, which provide a good balance between approximation quality and generation speed, while still allowing the exact output CDF to be determined and optimised for relative error.

Weight optimisation: Finally, the accurate approximation of weights requires a technique which is robust and computationally tractable, while also able to deal with weights spanning a huge dynamic range and long-tailed kernels with significant overlap. A high-precision least-squares approach is developed in Section III-C.

A. Accurate Alias Tables

Walker's alias method is an algorithm for sampling from a finite set of elements with an arbitrary probability of selecting each element [1], which can be considered as the problem of generating a random variate I over the integers $1..n$, with $\Pr[i = I] = p_i$. In the context of the PwCLT algorithm this is equivalent to selecting one of the component distributions according to the component weights.

The alias method uses two tables of n elements: a threshold table $t_1..t_n$ of probabilities in $[0,1]$, and an alternate index table $a_1..a_n$ in the range $[1..n]$. Sampling from the table requires two stages:

1) A uniform random integer $i \in 1..n$ is generated, selecting a pair (t_i, a_i) from the tables.
2) A uniform random real $y \in [0,1]$ is generated. If $y < t_i$ then the integer i is returned as the sample from I, otherwise the alternate index a_i is returned.

This method maps well into FPGAs as long as n is a power of two (allowing efficient generation of i), and as long as the tables required are similar to the size of typical LUT-RAMs or Block RAMs.

Previous uses of alias tables in hardware have performed the second step in fixed-point [3], [2], so the thresholds $t_1..t_n$ are rounded to w_t fractional bits, and the random integer y is simply w_t uniform random bits interpreted as a fixed-point number in the range $[0,1)$. However, this imposes a bound on the accuracy of the output distribution: if the thresholds t_i are rounded to the closest fixed-point number, then the absolute PMF accuracy will be around 2^{-w_t}, but the relative accuracy $(t_i - \text{round}(t_i))/t_i$ degrades as $t_i \to 0$. If $t_i < 2^{-w_t-1}$, then the closest fixed-point value is 0, resulting in a relative error of 1.

While it is possible to simply increase w_t, this becomes expensive in terms of storage. As i increases, corresponding to supporting larger output ranges, the probability of selecting each component decreases, with $t_i \propto \phi((i-1)\delta)$. For example, with triangular components and $\delta = 1$, $t_{8/\delta} = 1.05^{-13}$, and so to represent this threshold with a relative accuracy of 2^{-24} would require approximately $w_t \approx 67$. This gets much worse when moving further into the tails: 10σ requires $w_t \approx 93$ and 12σ requires $w_t \approx 124$, but a typical block-RAM is naturally only 36 bits wide, so multiple RAMs are needed.

Given the need to support relative error, the natural answer is to turn to floating-point thresholds. Usually floating-point operations are much more expensive than fixed-point, but in this situation there are two reasons why floating-point is very cheap:

- The random number y is the uniform distribution, rather than a more complicated distribution.
- The thresholds t_i are all compile-time constants which are non-negative in the range (0,1).

The first property allows efficient generation of the floating-point random numbers without using floating-point operations, while the second allow the floating-point range to be customised to the range of the thresholds.

Floating-point uniform numbers can be generated by using a wide fixed-point uniform generator $U_{fx}()$ to provide a random number u in the range [0,1), then converting it to floating-point. This requires a leading-one detector (lod) to find the most significant bit, a shifter to normalise the mantissa, rounding of the mantissa, followed by a possible re-normalisation. However, because the uniform bits used to determine the exponent are never used within the mantissa bits, it is possible to generate the two independently. In addition, rather than rounding the fractional part, it can be

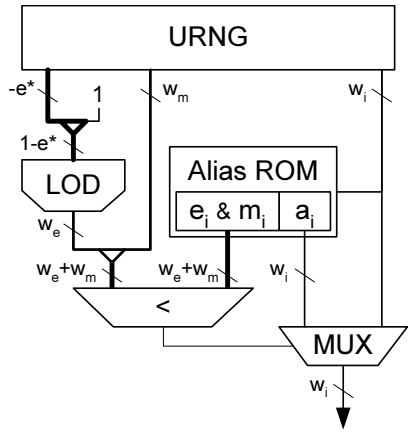

Figure 1. Hardware architecture of floating-point alias-table.

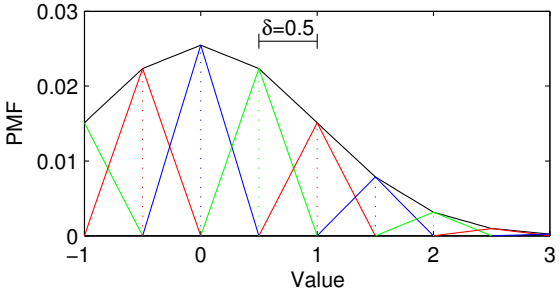

Figure 2. Approximation to the Gaussian with $\delta = 0.5$ and triangular component distributions.

generated directly as $w_m - 1$ uniform bits.

$$e_u = \text{lod}(U_{fx}()) \tag{6}$$

$$m_u = 0.5 + 2^{-w_m} \left\lfloor 2^{w_m - 1} U_{fx}() \right\rfloor \tag{7}$$

$$U_{fp} = m_u \times 2^{e_u} \tag{8}$$

The exponent e_u is a standard geometric variate, so given a particular floating-point threshold $2^{e_x} m_x$, the probability of a uniform floating-point number being lower than the threshold is exactly the same as the threshold.

The hardware architecture for the alias-table component is shown in Figure 1. Apart from the leading-one detector (LOD), the floating-point unit is exactly the same as the fixed-point detector used in previous work, but is able to use much smaller tables to achieve the same relative error. Note that the LOD can be arbitrarily pipelined without requiring additional matching registers, as there is no need to ensure that the uniform bits for the index, mantissa, and exponent are generated in the same cycle.

B. Central-Limit Component Distributions

The second architectural problem is deciding which component distributions can be generated efficiently in hardware. When matching arbitrary target distributions it is useful to be able to generate as many shapes as possible in order

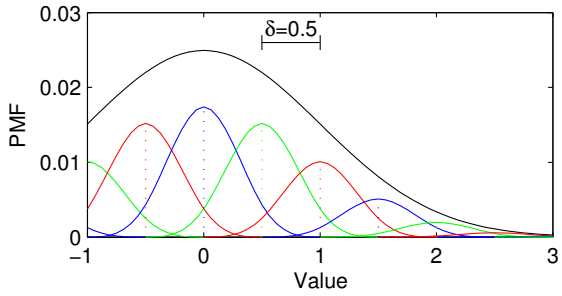

Figure 3. Approximation to the Gaussian with $\delta = 0.5$ and CLT_4 component distributions.

to match discontinuities in the PDF, leading to the trapezoid generator [3]. However, when attempting to match the known Gaussian distribution such flexibility is not needed, as it is know that there are no discontinuities, and that in fact the Gaussian CDF is very smooth.

Triangular component distributions can be formed as the sum of two uniform distributions, allowing a piecewise-linear PMF to be generated [2]. An advantage of using the triangle distribution as the component distribution is that if component i is a triangle offset from zero by δi, then the approximation at δi is exactly p_i:

$$a(x) = \sum_{i=1}^{n} [p_i f_T(x/\delta))] \tag{9}$$

$$a(\delta i) = p_i f_T(0) = p_i \tag{10}$$

In the range $[i\delta, (i+1)\delta]$ the approximation PDF will be linearly interpolated between p_i and p_{i+1}. A very low resolution approximation to the Gaussian is shown in Figure 2.

The triangular method is very simple, and makes it trivial to choose the approximation's PDF (or CDF) at regularly spaced points, but only allows the target PDF to be approximated using straight-line sections (or equivalently, the CDF can only be approximated using quadratics). Given enough triangles and a low enough δ the approximation can be made arbitrarily close, but achieving high accuracy requires a very large number of triangles, and hence a large amount of RAM.

The solution used here extends the Triangle approach to higher orders, so instead of adding just two uniform samples, k samples are added to produce the kernel distribution. Due to the CLT these kernels become closer to the Gaussian, and rapidly become very smooth. The distribution of these kernels can be described as:

$$CLT_k = \sum_{i=1}^{k} -1^i U_i, \qquad \text{where } U_1..U_n \sim U(0,1) \tag{11}$$

and has support over the range $(-k/2, k/2)$. The alternating summation is needed to ensure symmetry when working with finite-precision fixed-point, as a w_u bit fixed-point uniform sample actually ranges over $[0, 1-2^{w_u}]$ rather than (0,1). So for example, $CLT_2 \sim U_1 - U_2$, which is a discrete triangle distribution, is symmetric about zero. Each distribution CLT_k can be described as a piecewise distribution over k intervals with endpoints $-k/2, -k/2+1, ..., k/2-1, k/2$, and the CDF in each interval is described by a degree k polynomial. This closed-form distribution allows us to exactly recover the PMF or CDF of the PwCLT generator at any chosen point.

Assuming the use of LUT-Optimised uniform random generators [4], the cost of a triangle generator is $3w_u + 1$ LUT-FF pairs: $2w_u$ bits for the two w_u-bit uniform inputs, and one $w_u + 1$ bit subtractor to combine them. For even k the resource cost in LUT-FFs is then given by:

$$r(CLT_k) = \log_2(k) + w_u + 2r(CLT_{\lfloor k/2 \rfloor}) \tag{12}$$

$$r(CLT_2) = 3w_u + 1 \tag{13}$$

For large k this consumes significant area, far too much to use as a means of approximating the Gaussian distribution directly, but for $4 \leq k \leq 16$ it provides a cheap way of synthesising kernels which are very smooth - the kernel essentially allows approximation of a degree-k polynomial without any multiplications.

As with the triangle distribution, the CLT family can be spaced at equally spaced co-ordinates and used to approximate a distribution. But a big difference between $T = CLT_2$ and CLT_k with $k \geq 4$ is that the range of each kernel extends beyond the mid-point of its neighbour, so each kernel interacts with a large number of neighbours. This effect is shown in Figure 3, where each CLT_4 kernel overlaps four other kernels, resulting in a much smoother distribution than for the triangular approximation. This overlap means that the simple approach from the triangular kernel of choosing $p_i = t(i)$ can no longer be used. Instead a more complex form must be used, described in the next section.

C. Weight Selection

The two previous sections outlined the two architectural parts of the PwCLT generator: accurate alias-tables using fixed-point or floating-point to choose i according to p_i, and regularly spaced CLT_k kernels to provide the component distributions. This means that the only remaining decision made is to select p_i such that the approximation is sufficiently close to the target. Because the kernels overlap, we cannot perform purely local optimisations, as any change to one component's weight affects a large surrounding area of the approximated PMF.

The approach used here is to fit the CDF by solving an over-constrained weighted least-squares. A set of sample-points $\vec{x}$ are defined within the target range, along with the expected CDF $\vec{y} = \Pr[\vec{x} < T]$. The weighting is taken to be $\vec{w} = 1/\vec{y}$, to target relative error – as mentioned earlier, a given absolute error level is much worse if found in the tails rather than the median of the distribution. The problem is then stated as a least-squares problem:

$$\vec{p} = (\mathbf{A}\vec{w}) / (\vec{y}\vec{w})$$

$$\mathbf{A} = \begin{bmatrix} \vec{z}_{-n/2} & \dots & \vec{z}_{-1} & \vec{z}_0 & \vec{z}_1 & \dots & \vec{z}_{n/2} \end{bmatrix}$$

where $\vec{z}_i = \Pr[\vec{x}/\delta - i < CLT_k]$. Solving for $\vec{p}$ provides the probabilities required for each kernel.

Surprisingly, no extra constraint is needed to ensure that the kernel probabilities add up to one. In the specific case of the Gaussian, the solution naturally produces a valid selection of weights which can be immediately turned into an alias table. Early experiments actually incorporated a Lagrange multiplier to ensure this constraint, but it had no real impact on the solution, and appeared to make the problem less stable.

The main practical problem when solving this system is the numerical precision and range required. When creating Gaussian generators which push far into the tails the CDF gets extremely small, much lower than the relative precision of a double-precision number. For example at -10σ the target CDF is 2^{-76}, which is far smaller than the 53-bit resolution of double-precision.

The solution used for this paper is simply to use higher precision numbers and a very stable solver: a QR-decomposition solver based on NTL's arbitrary precision RR package was developed [5], and the least-squares problem is solved using 256-bit floating-point. This approach is excessive when creating the lower accuracy generators, but is required when pushing into the 10σ and 12σ ranges where double-precision solvers fail catastrophically.

IV. Parameter Selection

The PwCLT approach admits many possible configurations for the Gaussian generator. Independent parameters which can be tuned are:

- **Output precision (w_f):** The intrinsic resolution of the generator, given as the number of fractional bits in the output
- **Component distribution (k):** The number levels of aggregation used in the component generators, which controls component smoothness.
- **Number of components (n):** The number of component distributions with non-zero probability, which determines the table size required and the output range of the generators.

Each of these parameters has an effect on resource usage of a GRNG, but also affects the quality of the output.

In principle one could define desired accuracy requirements and attempt to find the lowest resource solution, but here a simpler approach is taken, and three different (notional) quality levels are chosen in terms of GRNG support and CDF accuracy.

PwCLT-8 generator is appropriate for most people working with Monte-Carlo simulations or Bit-Error-Rate simulations – it offers much improved performance when compared with existing high-quality generators in terms of area and quality, yet still provides the same high speed. The 11 fractional bit output is chosen to match those of the two previous quality-oriented generators [6], [7], but the cost of increasing or decreasing the fractional bits is very low: each additional output bit requires 7 extra LUT-FFs, so for example extending the fractional width from 11 to 24 requires just 91 extra LUT-FFs.

For people with stringent requirements, the **PwCLT-10** offers extremely high quality and range coverage, and is appropriate for very long-running simulations utilising thousands of parallel simulation cores. The resource increase over PwCLT-8 is not large, so it also represents a good tradeoff for designers wishing to future-proof their designs against potential increases in simulation scale.

The **PwCLT-12** is offered more as a curiosity – the range it provides is so high that no practical simulation that could be implemented using current FPGAs should be able to probe into the tails of its distribution. The chance of observing a sample outside the $\pm 12\sigma$ range is 10^{-33}. Even if one were able to use 10^6 FPGAs, each containing 10^6 GRNGs, running at 10GHz, the waiting time would still be of the order of 10^5 years before seeing a sample outside this range. The achieved quality is correspondingly high - within the "practical" $\pm 10\sigma$ range the relative CDF error is strictly below 10^{-10}.

V. Related Work

Many algorithms for GRNGs have been developed, with various tradeoffs in terms of memory consumption, uniform samples consumed per Gaussian output sample, and average computation per output sample [8]. While these algorithms were usually designed with CPUs in mind, many of them have been adapted to hardware implementation via FPGAs.

One of the classic methods of generating Gaussian samples is the Box-Muller method, which transforms two independent uniform variates U_1 and U_2 into two independent Gaussian variates G_1 and G_2:

$$G_1 = \sqrt{-2\log U_1}\cos(2\pi U_2) \quad G_2 = \sqrt{-2\log U_1}\sin(2\pi U_2) \tag{14}$$

There have been a number of reported implementations of the Box-Muller method in FPGAs with the most refined implementation provided by Lee et. al. [9]. Their implementation starts from the output requirements of 11 fractional bits and the ability to generate samples in the range $\pm 8.2\sigma$, and then carefully propagates these accuracy and range requirements backwards through the chain of operators. The resulting implementation guarantees a faithfully rounded transform while using the minimum width operations throughout, providing high quality while using less resources than previous Box-Muller implementations.

Another popular approach is the inversion method, which generates a Gaussian sample by inverting the CDF function – if the CDF (Φ) maps a Gaussian variate to a probability (i.e. uniform variate), then the inverse CDF ($\Phi^{-1}(x)$) should map a uniform variate to a Gaussian variate:

$$G = \Phi^{-1}(U) \tag{15}$$

As with the Box-Muller method, it is possible to specify the precision and range of the Gaussian output samples, then propagate this information back through a custom-built inversion pipeline. A highly-optimised implementation using non-uniform segmentation into a table of degree-1 polynomials has been developed [6], which uses fewer resources than the Box-Muller method to provide the same faithfully-rounded accuracy guarantees.

Other software algorithms adapted for FPGAs include the Ziggurat method [10], and the Monty-Python method [11]. These are rejection methods, meaning they are not guaranteed to produce a result every cycle, and also rely on a frequent fast-path plus infrequent slow-path. This makes them somewhat less efficient in hardware than software, and it can be complex for applications to deal with stall cycles when no random number is generated.

Two recent hardware-specific methods are the CLT-Corrector generator [12], and the Table-Hadamard generator [13]. The CLT-Corrector method is based on the same CLT_k type distributions used here, but only uses one component centred at zero, with a polynomial stretch to attempt to match the distribution to the Gaussian. The Table-Hadamard starts with many IID samples from an approximately Gaussian distribution, then adds them all up to produce a more Gaussian output. It is also designed to generate large vectors of Gaussians per cycle, rather than scalars – this makes it efficient in a resources-per-output sense, but it presents challenges in terms of congestion and fanout. Both methods provide a very good performance-area point, but are not explicitly aiming for very high quality in the same way as the Box-Muller and Inversion methods.

VI. Evaluation

The generators presented here are first compared in terms of quality with the two previous generators which focus on statistical quality: the Box-Muller by Lee et. al. [9], and the Inversion method from Cheung et. al. [6]. While we have mentioned other competing methods for GRNGs in FPGAs, they mainly focus on reducing resource requirements without considering quality, or only consider weak empirical quality metrics – we will consider these generators in terms of performance and resources later.

There are a number of polynomial coefficients needed in the Box-Muller and Inversion methods, not included in the two respective papers, so to be conservative we assume *all* stages are correctly rounded in those two methods. Under that assumption, we can recover the best-case PDF for the two generators, but this is deliberately an over-estimate of their quality in real hardware. In comparison, the PwCLT generators are evaluated in terms of the *exact* CDF of the register level hardware, with no approximations at all.

A. Relative CDF Error

This measures the maximum relative CDF error within a particular CDF range, defined in terms of the included range, or equivalently in terms of excluded tail probability. For example, the $\pm 8\sigma$ range corresponds to the probability range $[\Phi(-8)..\Phi(+8)] = [10^{-15.2}, 1 - 10^{-15.2}]$, and the probability of observing samples outside this range should be $2\Phi(-8) = 10^{-14.9}$. Given a range $\pm r\sigma$, the relative error metric for a generator X is defined as:

$$\max_{-r \le x \le 0} \frac{|\Pr[x \le X] - \Pr[x \le N_C]|}{\Pr[x \le N_C]} \tag{16}$$

Because all GRNGs considered here are exactly symmetric about zero by construction, it is sufficient to consider just the negative half of the CDF.

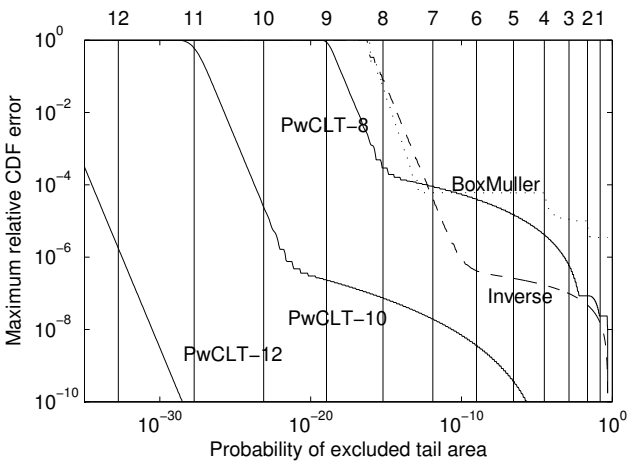

Figure 4. Maximum relative CDF error of the GRNGs over ranges of increasing excluded tail probability (decreasing σ range).

Figure 4 shows the relative error results for the different generators, with the point at which the error rises to 10^0 representing the point at which each generator's CDF falls to zero (i.e. the maximum output range of each GRNG). All the PwCLT generators have tails which extend to one more than their target σ, and all have less than 0.1% error within the target range. In comparison the Box-Muller and Inverse methods only just achieve their target 8σ range, and have significant error at that point. The PwCLT-8 method is generally better than the Box-Muller method, except for a small overlap at 7σ. The Inverse method appears significantly better than PwCLT-8 within 7σ, but this is due to the (known-optimistic) assumption of correct rounding.

B. Empirical Tests

Empirical tests examine the properties of finite-size samples of random numbers produced by the GRNG, and apply hypothesis tests. The hypothesis tests only produce p-values, which must then be classified as passing or failing according to some criterion. The method chosen here is to apply the tests for increasing sample sizes, and then label the first p-value below 10^{-6} as the sample-size at which the test clearly fails. Because RNGs near the point of outright failure sometimes "circle the drain", reporting consecutive suspicious p-values, the metric used here is the "last known good" sample-size, defined here as the last sample-size producing a value above 0.01.

For each test the sample sizes chosen are binary powers from 2^{16} up to 2^{96}. For sample sizes less than 2^{33} the samples were generated directly, by taking outputs from the GRNG. However, larger sizes are impractical to generate in a reasonable period of time; for example, 2^{96} would require around 10^{12} years at 1GHz. Instead the histograms are generated directly, using the BTPEC binomial algorithm [14] to exactly generate histogram counts according to the known probability of each output value. Generating histograms rather than individual samples should be equivalent, under the assumption that the source of uniform bits in hardware is

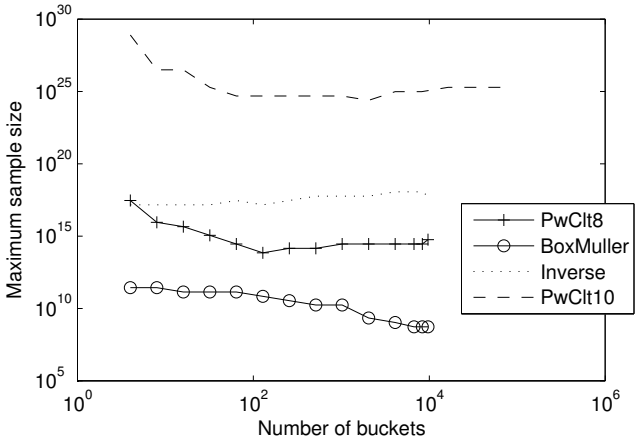

Figure 5. Maximum sample size before failure for χ^2 test with increasing numbers of buckets.

truly IID uniform. At some level that must not be true (unless the uniform source uses a non-deterministic entropy source), but in practice the modern methods available for generating uniform bits in hardware seem sufficiently IID; no statistically significant discrepancy could be found between the GRNG output samples and directly generated histograms for a sample size of 2^{40}.

The ability to pass the test is checked using a reference GRNG, which uses a very large double-precision alias table to implement the exact target probability distribution, and the PwCLT-12 generator. Both generators pass all three tests convincingly for a sample-size of 2^{96}, demonstrating that a sufficiently high-quality generator can pass the tests.

The χ^2 tests split the range up into equal probability buckets, taking into account the fixed-point nature of the range, then count the number of samples which end up in each bucket. The bucket counts are binary powers ranging from 16 to 2^{16}, which combined with the minimum sample-size of 2^{20} ensures that there are always at least 16 expected samples per bucket. Figure 5 shows the number of actual buckets achieved (which varies according to the output precision of each generator), along with the maximum sample size.

The CDF inaccuracy seen in the Box-Muller's relative error has a big effect on the empirical results, severely limiting the sample size that can be achieved. The PwCLT-8 can support sample sizes two orders of magnitude larger before failing, though it is still beaten by the idealised inversion method. The analytical quality of the PwCLT-10 is clearly shown in the achievable sample size, supporting a sample *six orders of magnitude* larger than inversion. PwCLT-12 is not shown because it passes all tests.

C. Resource Utilisation

Developing platform independent FPGA designs is challenging, as the low-level fabric changes between vendors and device generations. The PwCLT generators are not specific to any particular generation or family, but some of the specific choices, such as the number of components, are chosen to match current devices. The PwCLT-8 architecture in particular is tuned for contemporary architectures such as Virtex-6 and Stratix-III which allows LUTs to be efficiently configured as 128-bit ROMs.

GRNG	Slice	LEs	LUT	FF	RAM	MHz
PwCLT-8	104	348	291	333	0	741
	89	304	283	278	1	595
PwCLT-10	191	629	572	589	0	654
	144	531	579	507	1	591
PwCLT-12	214	707	677	651	1	600

Table I
RESOURCE USAGE OF PWCLT GRNGS IN VIRTEX-6 XC6VLX75T-3.

The area and performance of the three generators is given in Table I, giving the resource consumption in terms of low-level primitives. For PwCLT-8 and PwCLT-10 it is reasonable to use either LUT-ROMs or block-RAMs to hold the alias table, so both methods are considered. The performance results are post place-and-route results from Xilinx ISE 13.1 with a clock constraint of 600MHz. It should be emphasised that these are in fact post-place and route timing results, despite the relatively high clock rates. Because there are no multipliers involved, and all the carry-chains are relatively short, it is possible to achieve very high speeds, even without placement. In an actual application, such rates are unlikely, due both to the congestion of a typical design, and to other parts of the circuit using DSP and BRAM resources. However, it shows the intrinsic speed and efficiency of the generators, due to the careful selection of resources.

Comparing resource utilisation between GRNGs is often difficult, due to the lack of source code and differences between FPGA generations. To allow comparison with previous work, the PwCLT generators have been re-compiled for the older Virtex-4 devices, with the results shown in Table II. The Monty-Python and Ziggurat generators are elided, as they do not present a competitive tradeoff in either performance or quality.

Note that no device-specific resource optimisations or tuning were performed for the PwCLT Virtex-4 results, so there is significant scope for vendors to produce much more tightly optimised version which balance pipeline stages against specific architectural features. Because the Box-Muller and Table-Hadamard methods generate vectors of samples, the resource utilisation has been normalised to a single output.

In terms of resources, the PwCLT-8 generator is smaller than other approaches, requiring no block-RAMs or DSP blocks, and still requiring fewer slices. The exception to this is the Table-Hadamard generator, which is both much smaller, and has somewhat better quality. The big difference here is that the Table-Hadamard is a vector generator, while PwCLT-8 is a scalar generator. So the advantages of Table-Hadamard only apply if you need all 64 samples it generates per sample – if you need any fewer, then either the quality

GRNG Name	Resources			Samples per Sec	Output Range		Max Rel. CDF Err.		χ^2 Fail
	Slices	DSPs	BRAMs		w_f	$\pm\sigma$	8σ	10σ	
Box-Muller [9]	726	6	1.5	375	11	8.2	$10^{-1.1}$	1	10^{8}
Inversion [6]	487	2	2	371	11	8.2	$10^{-1.1}$	1	10^{17}
Table-Hadamard [13]	102	0	0	351	11	12+	$10^{-2.1}$	$10^{-1.3}$	10^{13}
CLT-Corrector [12]	420	1	0	220	11	6+	-	-	-
PwCLT-8	392	0	0	521	11	9.1	$10^{-3.3}$	1	10^{13}
PwCLT-10	776	0	0	358	16	11.4	$10^{-7.1}$	$10^{-5.2}$	10^{24}
	455	0	2	376					
PwCLT-12	576	0	3	321	20	13.2	$< 10^{-12}$	$< 10^{-12}$	$> 10^{29}$

Table II
COMPARISON BETWEEN PREVIOUS GRNGS AND THE PWCLT GRNGS IN THE VIRTEX-4 XC4VLX100 PART.

goes down due to fewer Hadamard levels, or the resources per generated output will increase. Because Table-Hadamard is a large, densely connected block it experiences significant congestion, but PwCLT-8 is relatively small, providing a 50% performance advantage.

When looking at PwCLT-10, there is just no competition in terms of quality from any known hardware GRNG – the achieved analytical and empirical quality is of a completely different order, despite having similar performance and resource requirements to the Box-Muller and Inversion, though again, Table-Hadamard is more resource efficient when generating vectors. Going further, the quality of the PwCLT-12 is so high it is difficult to actually quantify, even though the speed and resource usage is comparable to previous best in class generators.

VII. CONCLUSION

The PwCLT approach provides a highly efficient means of sampling from the Gaussian distribution in FPGAs, by using the combination of very accurate fixed and floating-point alias-tables, and smooth high order component kernels. The resulting generators are able to closely approximate the target Gaussian, even far out into the tails where probabilities become exceedingly small. Using a number of asymptotic and empirical tests, it is possible to establish at what sample size the generators would be expected to fail in practice. This allows a number of different PwCLT generators to be targeted at different quality levels: the PwCLT-8 generator which exceeds the range and quality of previous scalar approaches while using low resources; the PwCLT-10 generator which provides guaranteed quality far exceeding that of any previous approach with similar resource usage; and the PwCLT-12 which can provide quality suitable for decade-scale simulations spanning parallel resources across the planet.

The three generators proposed in this paper are made available online as VHDL source code, both to allow use in applications, and to allow designers of future GRNGs to know exactly what coefficients and parameters were chosen when constructing the generators: http://cas.ee.ic.ac.uk/people/dt10/research/rngs-fpga-normal.html

REFERENCES

[1] A. J. Walker, "An efficient method for generating discrete random variables with general distributions," *ACM Trans. Math. Software*, vol. 3, pp. 253–256, 1977.

[2] D. B. Thomas and W. Luk, "Non-uniform random number generation through piecewise linear approximations," *IET Computers and Digital Techniques*, vol. 1, no. 4, pp. 312–321, 2007.

[3] ——, "Efficient hardware generation of random variates with arbitrary distributions," in *Proc. FCCM*, 2006, pp. 57–66.

[4] ——, "High quality uniform random number generation through LUT optimised linear recurrences," in *Proc. FPT*. IEEE Computer Society, 2005, pp. 61–68.

[5] V. Shoup, "NTL: A library for doing number theory," http://www.shoup.net/ntl/.

[6] R. Cheung, D. Lee, W. Luk, and J. Villasenor, "Hardware generation of arbitrary random number distributions from uniform distributions via the inverson method," *IEEE Transactions on VLSI*, vol. 15, no. 8, pp. 952–962, 2007.

[7] D.-U. Lee, W. Luk, J. D. Villasenor, and P. Y. Cheung, "A Gaussian noise generator for hardware-based simulations," *IEEE Transactions On Computers*, vol. 53, no. 12, pp. 1523–1534, december 2004.

[8] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor, "Gaussian random number generators," *ACM Computing Surveys*, vol. 39, no. 4, p. 11, 2007.

[9] D. Lee, J. Villasenor, W. Luk, and P. Leong, "A hardware Gaussian noise generator using the box-muller method and its error analysis," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 659–671, 2006.

[10] G. L. Zhang, P. H. Leong, D.-U. Lee, J. D. Villasenor, R. C. Cheung, and W. Luk, "Ziggurat-based hardware Gaussian random number generator," in *Proc. FPL*. IEEE Computer Society Press, 2005, pp. 275–280.

[11] Y. Li, P. Chow, J. Jiang, M. Zhang, and S. Wei, "Software/hardware framework for generating parallel gaussian random numbers based on the monty python method," in *Proc. FPT*, 2012, pp. 190–197.

[12] J. S. Malik, J. N. Malik, A. Hemani, and N. D. Gohar, "Generating high tail accuracy gaussian random numbers in hardware using central limit theorem," in *Int. Conf. VLSI and SoC*, 2011, pp. 60–65.

[13] D. B. Thomas, "Parallel generation of gaussian random numbers using the table-hadamard transform," in *Proc. FCCM*, 2013.

[14] V. Kachitvichyanukul and B. W. Schmeiser, "Algorithm 678 btpec: sampling from the binomial distribution," *ACM Trans. Math. Softw.*, vol. 15, no. 4, pp. 394–397, 1989.